

SM-KIT/M

Assembler-Tool für Commodore-Computer

Scherbaumstraße 29
8000 München 83

Bedienungsanleitung
SM – KIT / M

Scherbaumstraße 29
8000 München 83

Telefon: 0 89/6 37 12 11
Aut. Auftragsannahme
Telefon: 0 89/6 70 58 13

Druck: Reinz Druck GmbH

Inhaltsverzeichnis

Vorwort 2
Einbauanleitung 5
Allgemeines 7
Ein- und Ausschalten 7
Funktionsübersicht 8
Bemerkungen zum Beispiel 9

1. Assembler 10
2. Disassembler 13
3. Umschaltung Hex-Dez 15
4. Aus- und Eingabe von Bytes 17
5. Aus- und Eingabe von Adressen 19
6. Aus- und Eingabe von Zeichen 21
7. Blocktransport 23
8. Definition des Programmbereichs 25
9. Insert / Delete 28
10. Find 33
11. Trace 38
12. Break-Point 49
13. Ausgabe von BASIC-Zeilenadressen 51
14. Einbau von REM-Routinen 53

ROM-Adressen für Find 55

Vorwort

SM-Kit/M ist ein Programmier-, Test-Unterstützungs-, Fehlersuch- und Programm-Analyse-Werkzeug für Maschinenprogrammierer auf CBM-Rechnern. SM-Kit/M benötigt SM-Kit/B.

Wenn Sie die im folgenden beschriebenen Eigenschaften mit anderen Tools vergleichen, werden Sie verstehen, warum nur der SM-Kit das SM-Zeichen tragen darf (und von Commodore empfohlen ist).

Zwischen Hex- und Dez-Anhängern liegen Welten und oft genug entscheidet das verwendete Zahlensystem, ob ein Tool eingesetzt wird oder nicht. SM-Kit schaltet durch den leisen Wink mit dem '$' um - so oft Sie wollen.

Assemblieren und Disassemblieren von und mit **Mnemos, Bytes, Adressen und Zeichen** geht so einfach und automatisch, daß Sie bald vergessen haben, daß Ihr Computer vorher davon nichts wissen wollte:

Er macht nun sogar aus einer Notiz wie **lda(5y** einen anständigen Befehl: **lda (5),y** . Und die Zieladresse eines **Branch** dürfen Sie ruhig absolut angeben **659 bne 652** , er nimmt das Absolut nicht krumm, sondern rechnet es relativ: **659 bne -9 652.**

Daß SM-Kit **Blocktransport** beherrscht, wollen wir nicht ausführen.

Wichtiger ist da schon, daß er innerhalb eines Programmblocks beliebig viele **Bytes einfügen oder löschen** kann - wenn er will. Und er will **nicht,** wenn Sie **Zieladressen von Sprüngen löschen** würden, oder wenn Sie bezüglich **Branches** den Bogen **überspannen** würden - beim Einfügen. Dafür zeigt er aber alle Sprünge, die für Sie zum Problem würden wenn er nicht aufgepaßt hätte.

Vorwort

Was Sie immer schon finden wollten, sucht er für Sie: Z.B. alle schreibenden Zugriffe auf die Zero-Page-Zellen 180-200 oder alle JuMPs in den Bereich des Bildschirm-Editors oder wo die Zelle 31 als Argument einer indirekt indizierten Leseoperation verwendet wird. Und für all dies geben Sie einmal in einer Tabelle die Programm-Bereiche vor, die er durchsuchen soll (Die Tabelle der ROM-Bereiche finden Sie im SM-Kit/M-Handbuch und die Beschreibung des ROM-Inhalts im SM-ROM-Listing)

Manchmal müssen Sie aber etwas finden, das Sie selbst gemacht haben: einen **Fehler.** Dann lassen Sie sich vom **Trace** Ihr Programm Befehl für Befehl vorführen, lassen sich jedesmal alle Register-Inhalte ausgeben und die Status-Bits zeigen. **Ändern** können Sie alles, wenn es Ihrem Programm zur richtigen Entscheidung verhilft.

Falls Sie der Trace langweilt, weil er schon wieder die ROM-Routine zeigt, für deren Fehler Sie nicht verantwortlich sind, lassen Sie ihn ganz schnell bis in Ihr Programm zurück laufen - in **real-time.** Und wenn Sie eine lange Schleife Ihres Programms nach dem dritten Durchlauf nicht mehr im Verdacht haben, lassen Sie ihn ohne Ausgabe bis hinter die Schleife laufen: **überwachter Lauf ohne Ausgabe.**

Das Schönste kommt noch: Was Sie auf dem Bildschirm über der ersten Trace-Zeile an Disassembler-Ausgaben definiert haben, wird bei jedem Schritt auf den neuesten Stand gebracht (Bytes, Adressen, Zeichen).

Damit aber nicht genug: Nach jedem Schritt können Sie mit dem Cursor nach oben gehen und andere Ausgaben definieren oder Vorbesetzungen machen oder nur kurz mal was anschauen oder ...

Vorwort

Wenn Sie mit dem Trace nicht einfach mitten in einer Routine beginnen können, z.B. weil Sie von BASIC aus durch SYS gerufen wird, setzen Sie einfach einen **BReaK-Point** . Sobald der Prozessor auf einen Break läuft, restauriert der Trace den Code an dieser Stelle und wartet auf Ihr Einverständnis zum Weiterlauf.

Der Trace teilt **Ihrem Programm** eine **eigene Zero-Page** und einen **eigenen Stack** zu. Und er frägt Sie vorher nach dem **Austauschbereich.** Und Sie können in dieser Pseudo-Zero--Page und dem Pseudo-Stack nach Fehlern suchen oder für Bildschirmausgaben die Cursor-Zeiger manipulieren.

Wenn Sie jetzt wissen, daß Sie sich immer schon so etwas gewünscht haben, sagen wir Ihnen noch, daß **alle Ausgaben** (Disassembler, Find, Trace) **auch auf Drucker oder Floppy** möglich sind - wie bei SM-Kit/B von der Hardcopy-Möglichkeit ganz zu schweigen.

Wir wünschen Ihnen viel Spaß und Erfolg mit den SM-Kits.

München im Oktober 1981

SM Softwareverbund Microcomputer GmbH
Geschäftsbereich System-Software
Dipl.-Ing. H. Schießl

Einbau des SM-Kit/M

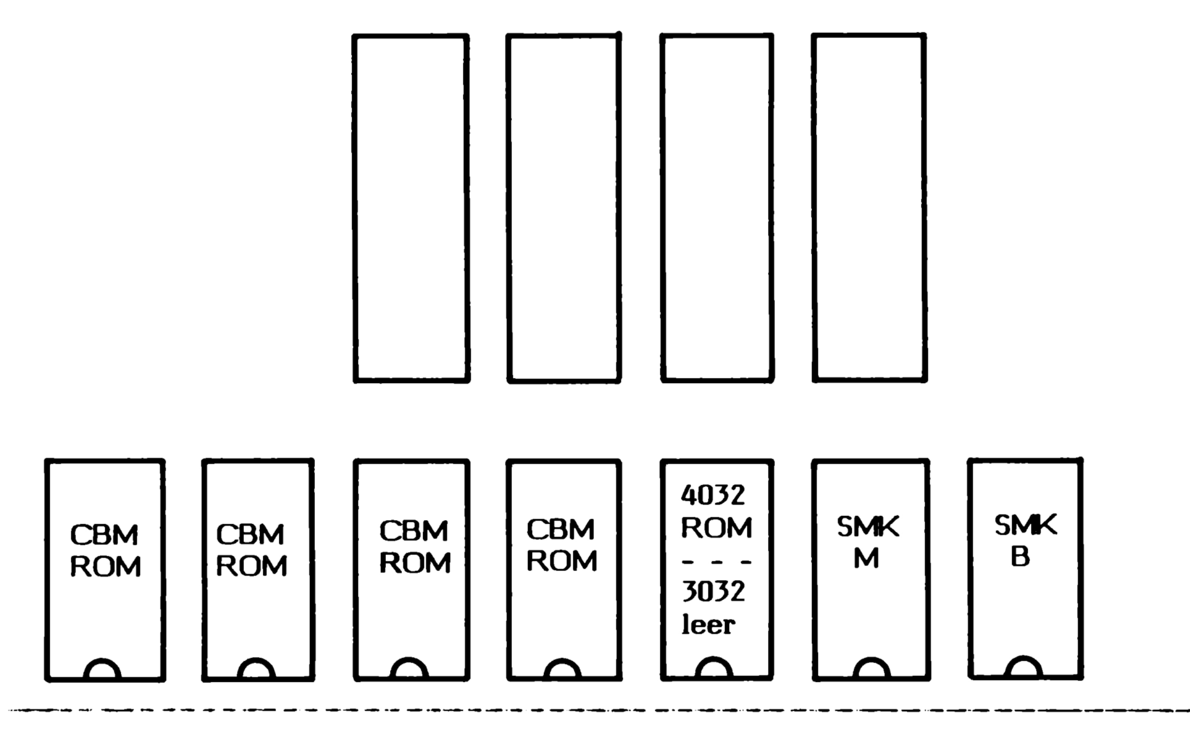
Der Baustein (EPROM) wird in den vorletzten (s. Skizze) nicht von Betriebssystem-ROMs belegten Sockel gesteckt. **Schalten Sie dazu den Rechner aus** und berühren Sie ein geerdetes Metallteil, ehe Sie den EPROM anfassen. Vermeiden Sie in jedem Fall eine Berührung der Beinchen mit den Fingern.

Der EPROM hat an einem Ende eine kleine Vertiefung (Nase). Diese Codierung muß in dieselbe Richtung zeigen, wie die von allen anderen Bausteinen, bzw. wie in der Skizze angedeutet.

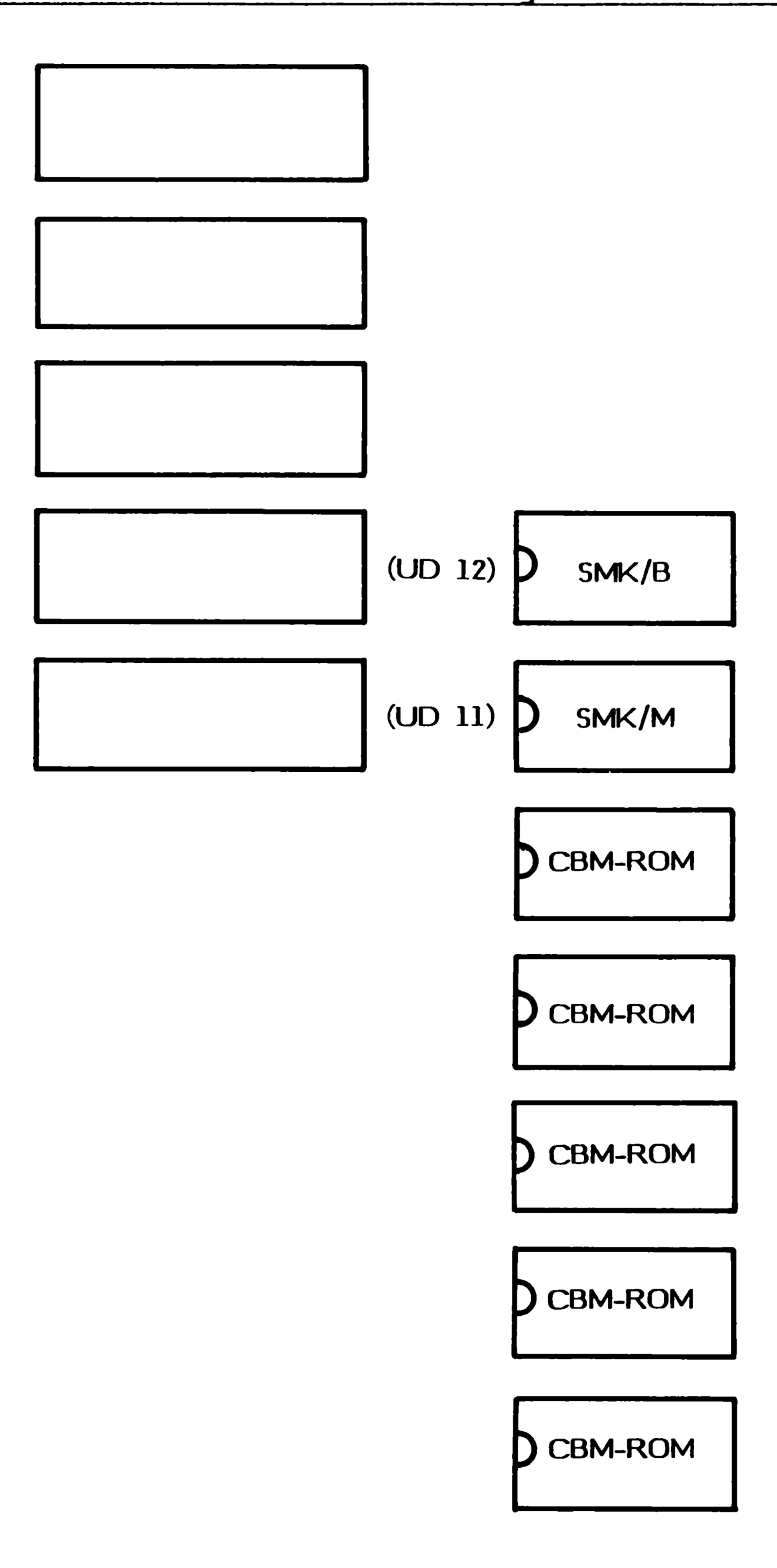
Wenn die Nase in die falsche Richtung zeigt, kann der Baustein oder Ihr Rechner Schaden nehmen.

Wenn sich nach dem Einschalten der Rechner nicht normal meldet, schalten Sie sofort wieder ab und überprüfen Sie den Einbau des Bausteins.

Lagezkizze für 3032/4032 mit kleinem Bildschirm



Lagezkizze für 8032/4032 mit großem Bildschirm



Allgemeines

SMK/M2.0 ist nur zusammen mit dem SMK/B2.0 für BASIC-Programmierung funktionsfähig. Es belegt den Adress-Bereich $A000-$AFFF.

Für die Rechner-Serie 8000/4000 (BASIC-4) existiert die Version 84-SMK.. . Sie funktioniert sowohl auf dem 4000-er mit dem kleinen als auch mit dem großen Bildschirm.

Für die Serie 3000 (BASIC-3) existiert die Version 30-SMK...

Einschalten: sys 40960

Meldung: smk-m2.?
smk-b2.?
ready.

Ausschalten: .x

Alle Eingaben werden auf den Bildschirm geschrieben und mit RETURN übergeben. Der für SM-Kit-Kommandos charakteristische Punkt '.' muß **immer in der ersten Spalte** der Bildschirmzeile stehen! Er funktioniert nicht an der ersten Stelle der zweiten Hälfte einer 80-Zeichenzeile, bitte beachten Sie dies bei Rechnern der Typen 4000 und 3000.

Die Ausgabe läuft, solange die RETURN-Taste niedergehalten wird. Die Disassembler-, Byte-, Adressen- und Zeichenausgabe stoppt nach 24 Zeilen. Dauerausgabe wird durch Einrasten von SHIFT LOCK bei gedrückter RETURN-Taste bewirkt. Ausgabe auf Drucker oder Floppy wird (wie bei SMK/B2.0) durch .- bzw. .-la ein- und durch .# wieder ausgeschaltet.

Zur Syntax:

ADR, BYTE, ...	steht symbolisch für eine konkrete Adresse, ein konkretes Byte usw. Im Dezimalmodus können hier bei der Eingabe auch Variablennamen verwendet werden.
a, b, c,..., $	bedeutet, daß dieses Zeichen (ohne SHIFT!) einzugeben ist.

Funktionen:

1. ASSEMBLER	. ADR MNEMO,OPERAND
2. DISASSEMBLER	. ADR
3. HEX-DEZ	. $, $ ADR, ADR $
4. Bytes	. ADR b (BYTE,BYTE,..)
5. Adressen	. ADR a (ADR)
6. Zeichen	. ADR ´ (ZEICHEN ..)
7. Blocktransport	. t ADR, ADR, ADR
8. Programmbereich	. p ADR, ADR
9. INSERT/DELETE	. i ADR, n bzw -n
10. FIND	. f(ADR),CODE:BEREICH
11. TRACE	. > (Pfeil nach rechts)
12. BREAK-Point	. b ADR
13. BASIC-Zeil.-Adr	. z ZNR
14. REM-Routinen	. tp,z ZNR

Bemerkungen zum SM-Kit/M Beispiel

Jeder Abschnitt dieser Beschreibung zeigt die Anwendung der dort besprochenen Befehle an einem Beispiel-Maschinenprogramm. Dieses schreibt bei Ausführung auf den Bildschirm eine (zwei) kurze Textzeile(n). Der Benutzer kann also die Wirkung (z.B. beim TRACE-Lauf) auf dem Bildschirm verfolgen.

Das Beispiel wurde so gewählt, daß ohne Tricks möglichst viele Funktionen des Maschinenteils des 'SM-KIT-M' gezeigt werden können. Das Beipiel selbst steht im Kassettenpuffer (#1). Der Beginner sollte in jedem Falle mit dem Abschnitt 1 dieser Beschreibung beginnen, da hier das erwähnte Beispiel besprochen wird und gezeigt wird, wie es in den Rechner eingegeben werden kann.

Da einige Befehle von 'SM-KIT-M' nicht nur die Eingabezeile (in der der Cursor gerade steht) sondern auch den weiteren Bildschirminhalt verändern, ist jeweils die Eingabe und die Rechnerantwort nach der Eingabe wiedergegeben. Auf diese Weise kann mit Hilfe dieses Textes die richtige Bedienung leicht kontrolliert werden.

Als **Aufmunterung für Beginner** sei bemerkt:

Erwarten Sie nicht von sich, daß Sie nach **Durchlesen** dieser Anleitung schon alle Befehle beherrschen. Sie werden erst nach einiger **Übung** den vollen Umfang aller Möglichkeiten des SM-KIT/M kennen **und** schätzen lernen!

1. Assembler

. ADR BEFEHL OPERAND

gibt den Befehl ab der angegebenen Adresse ADR ein und disassembliert den eingegebenen Kode zur Kontrolle sofort wieder. Dann wird in der Folgezeile die nächste Adresse automatisch ausgegeben und der Rest der Zeile gelöscht, um unerwünschte Eingaben zu verhindern. Hierzu ist noch eine Anmerkung zu machen:

Der Inhalt der Zelle **999** entscheidet darüber, in welcher Form die Vorgabe einer Folgezeile nach einer Eingabe (oder Ausgabe) erfolgt.

Es gibt drei Möglichkeiten :

1. Inhalt ist **gleich** 255	es wird nichts vorgegeben
2. Inhalt ist **größer** 127	nur die Adresse vorgeben der Rest bleibt stehen
3. Inhalt ist **kleiner** 128	Folgezeile wird gelöscht

Durch Abändern des Inhaltes von 999 (durch .999b WERT) kann die Vorgabeart jederzeit abgeändert werden.

Nach dem Disassemblieren ist immer Vorgabeart 3 eingestellt.

Für die Befehlseingabe gilt die übliche 6502-Syntax. Die ACCU-Schiebebefehle ASL, LSR, ROR und ROL, sind **ohne** den in der 6502 Syntax üblichen Folgebuchstaben A einzugeben. Auch beim Disassemblieren wird der Buchstabe A nicht ausgegeben.

Leerstellen zwischen Adresse, Befehlswort und Operand sind im allgemeinen überflüssig, aber zulässig. Sie sind im HEX-Modus notwendig, wenn andernfalls die Trennstelle nicht eindeutig erkennbar ist.

Bei relativen Sprungbefehlen nach rückwärts kann der Operand negativ (-128,...,-1) oder positiv (128,...,255) eingegeben werden; ferner kann bei diesen Befehlen auch die Zieladresse anstelle des Operanden eingegeben werden, worauf die Umrechnung automatisch erfolgt (in der Page 0 ist das natürlich nicht möglich).

In gewissen Grenzen werden Eingabefehler (z.B. fehlende Klammern usw.) korrigiert. Wenn nicht durch ? und Wiederholung der Adressenausgabe eine unverständliche Eingabe signalisiert wird, gibt die disassemblierte Ausgabe das Ergebnis der Eingabe wieder.

Beispiel

Bitte tippen Sie jedes Zeichen des folgenden Beispiels ein. Bitte beachten Sie, daß der Punkt immer in der ersten Spalte stehen muß! Leerstellen können ignoriert werden.

```
.   634  'SMK/M2.-BEISPIEL
.   650  ldy    #15
.   652  lda    634,y
.   655  sta 33408,y
.   658  dey
.   659  bpl   652
.   661  rts
.   662  brk
```

In Zeile 659 könnte als Adresse wahlweise entweder die wirkliche Zieladresse oder die relative positive oder negative Sprungweite angegeben werden, die Umrechnung erfolgt jeweils automatisch.

---

1. Assembler

Soll an irgendeiner Stelle die Assembler-Eingabe abgebrochen werden, so genügt es, hinter der Adresse einen Doppelpunkt einzugeben, der Rest der Zeile braucht nicht gelöscht zu werden. SMK/M2.0 restauriert nach RETURN den ursprünglichen Speicherinhalt, er übernimmt nicht den momentan nach dem Doppelpunkt auf dem Bildschirm sichtbaren Inhalt.

Beispiel:

Gehen Sie mit dem Cursor in Zeile 662 und geben dort nach 662 einen Doppelpunkt und z.B. STA33408,Y ein, dann drücken Sie RETURN. SMK/M stellt wie Sie sehen den alten Inhalt wieder her.

Eingabe:
.662:sta33408,y (RETURN)

Antwort:
.662 brk

Wenn Sie wollen, können Sie nun dieses Beispiel mit Hilfe von SMK/B2.0 abspeichern.

Eingabe:
.s"1:name",a634-663

Wie bei .S des Bildschirmmonitors wird ab einschließlich der angegebenen Anfangsadresse bis ausschließlich der Endadresse abgespeichert.

Das Laden dieses Beispiels (wie bei SMK/B erläutert) geschieht durch die Eingabe:

.l"1:name";

Die Wirkung dieses Beipielprogramms können Sie durch den Aufruf **SYS 650** auf dem Bildschirm verfolgen: es wird der Text ab Adresse 634 auf den Bildschirm geschrieben.

---

2. Disassembler

. ADR

listet ein Maschinenprogramm ab Adresse ADR. Die Ausgabe wird unterbrochen durch Loslassen der RETURN-Taste, sonst nach 24 Zeilen.

Die Operanden der relativen Sprungbefehle BEQ, BNE usw. werden im Dezimalmodus in den Bereich (-128,...,127) transformiert; dahinter werden in RVS-Darstellung die drei letzten Stellen der Zieladresse ausgegeben.

Trifft der Disassembler auf einen nicht übersetzbaren Kode, so schaltet er auf Byte-Ausgabe um, gibt ein Byte aus und stoppt, falls nicht durch SHIFT (oder SHIFT-LOCK) Dauerausgabe erzwungen wurde.

Bitte beachten Sie, daß bei den ACCU-Schiebebefehlen (nämlich ASL,LSR,ROR und ROL) der in der 6502 Syntax übliche Folgebuchstabe A beim Disassemblieren nicht ausgegeben wird.

Beispiel:

. 634

eingeben, dann **kurz** RETURN drücken, es erscheint in derselben Zeile die Textausgabe des Disassemblers (s. Ausgabe von Zeichen).

. 634 'SMK/M2.-BEISPIEL

Nun bitte den Apostroph in der Folgezeile nach (. 650) mit DELETE löschen und Taste RETURN kurze Zeit niederhalten, nun erscheinen bis zu 24 Zeilen Ausgabe.

2. Disassembler

```
. 650 ldy  #15
. 652 lda  634,y
. 655 sta 33408,y
. 658 dey
. 659 bpl   -9   652
. 661 rts
. 662 brk
```

Durch Einrasten der Taste SHIFT-LOCK kurz nachdem RETURN gedrückt wurde, erfolgt solange Dauerausgabe bis diese Taste wieder ausgerastet ist oder die SHIFT-Taste losgelassen wird.

Beachten Sie die Ausgabe des Operanden des relativen Sprungbefehls (Zeile 659), es werden die vorzeichenbehaftete relative Sprungweite und die letzten drei Stellen der Zieladresse (in RVS-Darstellung) ausgegeben. Trifft der Disassembler auf einen nicht übersetzbaren Kode, so schaltet er auf Byte-Ausgabe um, gibt ein Byte aus und stoppt, falls nicht SHIFT eingerastet oder niedergehalten ist. In diesem Fall erfolgt Dauerausgabe, hierbei kann natürlich nicht garantiert werden, daß der Disassembler richtig weiter arbeitet, Byte-Folgen (z.B. Datenbereiche) können zu unsinnigen Befehlsausgaben führen. Weiterhin kann nicht garantiert werden, daß sich SM-KIT bei wiederbeginnendem Befehlsbereich wieder richtig einfädelt.

3. Umschaltung Hex-Dez

Für die Ein- und Ausgabe von Bytes, Adressen und Operanden kann zwischen Dezimal- und Hexagesimaldarstellung frei gewählt werden. Nach dem Einschalten des SM-Kits ist der Dezimalmodus aktiv.

Folgende Umschaltmöglichkeiten stehen zur Verfügung:

.ADR $	schaltet jeweils in den anderen Modus um und gibt die Adresse adr in der nächsten Zeile neu aus. Diese Funktion kann deshalb auch zur Umrechnung verwendet werden.
. $ ADRh	schaltet den HEX-Modus ein.
. $	ohne Adressenangabe schaltet den Dezimalmodus ein.

Im HEX-Modus muß das $-Zeichen nicht jedesmal eingegeben werden. Dies ist lediglich bei Adressen am Zeilenanfang notwendig, die mit einem Buchstaben (A,...,F) beginnen, weil sonst der Buchstabe als Kommando interpretiert wird. Die automatische Adressenausgabe gibt deshalb am Zeilenanfang immer das $-Zeichen mit aus.

3. Umschaltung Hex-Dez

Beispiel:

(jeweils die 1. Zeile wird eingetippt, die
die zweite Zeile ist die Rechner-Ausgabe)

. 634$	Eingabe einer Dez-Zahl
. $ 27a	Ausgabe einer Hex-Zahl
. $ 27a	Hex-Modus einschalten
. $ 27b	Ausgabe
	Wenn hinter $27a nichts steht, wird **disassembliert,** wenn etwas steht, wird **assembliert,** wobei Operanden als Hex-Zahlen interpretiert werden.
. $	Dezimal-Modus einschalten
. 635	Ausgabe der letzten Adresse (hier $27b)
. $27a$	Eingabe einer Hex-Zahl und
. 634	Umrechnung in Dez-Zahl

4. Aus- und Eingabe von Bytes

Die angegebenen Anzahlen gelten für die Rechner 4032/3032, für Rechner der Serie 8000 sind die Werte in Klammern angegeben.

. ADR b

gibt die ersten 5 (10) Bytes (im Dezimalmodus) bzw. die ersten 8 (16) Bytes (im HEX-Modus) ab Adresse adr aus.

. ADR b BYTE BYTE ...

gibt die hinter b angegebenen Bytes ab Adresse ADR ein. Die Bytes sind durch Leerstellen getrennt anzugeben. Die Anzahl der Bytes ist bei der Eingabe nur durch die Zeilenlänge begrenzt. Nach der Übergabe durch RETURN werden die eingegebenen Bytes zur Kontrolle sofort wieder ausgegeben.

Anmerkung:

Nach RETURN wird die Folgezeile in derjenigen Vorgabeart ausgegeben, die vorher eingestellt war. Es gibt vier Vorgabearten:

.ADR	Disassemblieren
.ADR a	Aus-(Ein-)gabe von Adressen
.ADR b	Aus-(Ein-)gabe von Bytes
.ADR ´	Aus-(Ein-)gabe von Zeichen

Das hinter ADR stehende Zeichen, das die Ausgabeart bestimmt, bleibt also solange wirksam, bis es vom Bediener abgeändert wird.

4. Aus- und Eingabe von Bytes

Beispiel:

Die angegebenen Anzahlen gelten für die Rechner 4032/3032, für Rechner der Serie 8000 sind die Werte in Klammern angegeben.

Ausgabe von jeweils 5 (10) Bytes (im Dezimalmodus)

```
. 634b              ( eintippen !)
```

Antwort:

```
. 634   b   83   77   45   75   84
. 639   b   47   77   47   66   69
. 644   b   73   83   80   73   69
```

Ausgabe von jeweils 8 (16) Bytes (im HEX-Modus)

```
.$27a b             ( eintippen !)
```

Bitte beachten Sie, daß zwischen ´a´ und ´b´
Space erforderlich ist !)

Antwort:

```
. $ 27a    b   53 4d 2d 4b 54 2f 4d 2f
. $ 282    b   42 45 49 53 50 49 45 4c
. $ 28a    b   a0 0f b9 7a 02 99 b8 81
. $ 292    b   88 10 f7 60 00 ff ff ff
```

Die Anzahl der Bytes bei der Eingabe ist nur durch die Zeilenlänge (=80) begrenzt.

---

5. Aus- und Eingabe von Adressen

. ADR a

interpretiert die beiden Bytes in adr und adr+1 als L- und H-Byte einer Adresse und gibt diese Adresse aus.

. ADR a ADR1

gibt die Adresse ADR1 ab Adresse ADR ein.

Beispiel für Ausgabe:

.44a (eintippen und RETURN niederhalten)

Bitte beachten Sie, daß bei Ihnen auch andere Adressen auf dem Bildschirm erscheinen können!

```
.     42  a 17152
.     44  a 17873
.     46  a 18598
.     48  a 19238
.     50  a  1040
.     52  a 27141
.     ...........
```

5. Aus- und Eingabe von Adressen

Eingabe von Adressen

Wenn Sie den Wunsch haben, den Speicher oberhalb einer bestimmten Adresse zu schützen, so muß in die Zero-Page-Adressen 52/53 das Low- und High-Byte dieser Adresse gebracht werden. Bisher mußten Sie also die gewünschte Adresse in zwei Bytes zerlegt werden und dann durch POKE nach 52/53 bringen. Diese gesamte Funktion übernimmt jetzt SMK/M. Es soll z.B. in 52/53 die Adresse 24576 geschrieben werden.

Eingabe:

```
.   52  a 24576
```

Antwort:

```
.     52  a 24576
.     54  a
```

Ob in Zelle 52,53 wirklich die Adresse 24576 übernommen wurde, kann leicht durch die Byte-Ausgabe überprüft werden.

Eingabe:

```
.     52  b
```

Antwort:

```
.     52  b     0  96 135 255 135
.     57  b
```

In den Zellen 52,53 stehen nun also die beiden Bytes 0 und 96 (0+256*96 ergibt 24576).

6. Aus- und Eingabe von Zeichen

. ADR ´

gibt die den ersten 8 (16) Bytes ab Adresse ADR entsprechenden Zeichen aus. Nicht druckbare Zeichen werden als Zeichen in RVS-Darstellung ausgegeben.

. ADR ´zzzzz...

gibt die Kodes der durch zzzzz... symbolisierten Zeichen ein. Die Zahl der auf einmal eingebbaren Zeichen ist nur durch die Zeilenlänge (=80) begrenzt.

Vorsicht: Nicht-druckbare Kodes können auf diesem Weg nicht eingegeben werden, da die RVS-Darstellung bei der Übernahme in den Eingabepuffer ignoriert wird! Nicht druckbare Kodes sind deshalb immer als Bytes einzugeben.

6. Aus- und Eingabe von Zeichen

Beispiele:

Zeichen-Ausgabe:

.634´ (eintippen und **kurz** RETURN drücken)

Antwort:
. 634 ´SMK/M2.-BEISPIEL

Zeichen-Eingabe:

.826´zeichenausgabe 1

Antwort:

.826´zeichenausgabe 1
.842´

Die Eingabe kann kontrolliert werden durch :
.826´

Bitte beachten Sie, daß führende und endende Blanks nicht übergeben werden:

.826´ zeichenausgabe

Antwort:
.826 ´zeichenausgabe
.840 ´

7. Blocktransport

. t ADR1, ADR2, ADR3

kopiert den Inhalt des Bereichs (ADR1, ADR2) unverändert in den Bereich (ADR3, ...). Die fehlende Adresse ADR4 wird der Eingabe hinzugefügt.

.t ADR1, ADR2, , ADR4

kopiert den Inhalt des Bereichs (ADR1, ADR2) unverändert in den Bereich (..., ADR4). Die fehlende Adresse ADR3 wird in die Eingabe eingefügt.

Adressenbereiche gelten immer einschließlich Anfangsadresse, aber ausschließlich Endadresse!

Beispiel:

Der Textteil des Beipiels (Bereich von 634 bis 650) soll unverändert um 50 Speicherplätze weitertransportiert werden. (Der alte Bereich bleibt hierbei erhalten.)

Eingabe: (die Endadresse kann wegelassen werden)
.t634,650,684

Antwort:
.t 634, 650, 684, 700

Kontrolle ist leicht durch die Eingaben:
.634´ RETURN und .684´ RETURN möglich.

Antwort:
. 634 ´SMK/M2.-BEISPIEL
bzw.
. 684 ´SMK/M2.-BEISPIEL

7. Blocktransport

Um zu zeigen, daß auch die eventuell fehlende neue Anfangsadresse selbständig errechnet wird, soll das vollständige Beispiel so transportiert werden, daß es nach der Verschiebung bei der Zelle 800 endet.

Eingabe:

```
.t634,662,,800              ( zwei Kommas nach 662! )
```

Antwort:

```
.t    634,   662,   772,   800
.
```

Die Verschiebung (ohne jede Änderung!) kann leicht mit dem Disassemblerbefehl .772´ bzw. .788 kontrolliert werden:

```
.    772  'SMK/M2.-BEISPIEL
.    788  ldy    #15
.    790  lda    634,y
.    793  sta    666,y
.    796  dey
.    797  bpl     -9    790
.    799  rts
.    800  b  255
.    801
```

8. Definition des Programmbereichs

Bei den Funktionen INSERT/DELETE muß der Rechner wissen, wo das Programm beginnt und endet. Diese Information erhält er mit dem Kommando:

. p ADR1, ADR2

Wird der Bereich durch INSERT verändert, so wird die Endadresse automatisch nachgeführt; nicht so jedoch, wenn das Programm z.B. mit Hilfe des Assemblers fortgeschrieben wird.

. p

gibt die augenblicklichen Werte der Adressen aus.

. p ADR1

sucht das vermutliche Ende des Programms, indem als zweite Adresse die Adresse nach ADR1 ausgegeben wird, wo erstmals drei Kodes 170 unmittelbar aufeinander folgen. (Bei der RAM-Prüfung im Einschalt-Reset wird der gesamte RAM-Speicher mit diesem Kode - Bitmuster 10101010 - vollgeschrieben.) Wird keine solche Stelle gefunden, so wird 32768 (dezimal) bzw.$8000 (hex) ausgegeben.

8. Definition des Programmbereiches

Beispiel:

Nach RESET des Rechners (nach dem Einschalten) ist der Programmbereich noch undefiniert, man erhält dann nach der Eingabe:

.p

die Antwort:

.p 65535, 65535

Steht das Beispiel im Kassettenpuffer#1, so erhält man nach der Eingabe:

.p 634

die Antwort:

.p 634, 663

Merke: Hat man, nachdem man den Programmbereich richtig definiert hat, durch INSERT/DELETE ein Maschinenprogramm verschoben, so erhält man nur nach .p (ohne Adresse!) den nachgeführten Speicherbereich.

8. Definition des Programmbereiches

Nach der Eingabe (.p Anfangsadresse) erhält man jedoch als Antwort diejenige Adresse, wo nach der Anfangsadresse erstmals drei Kodes 170 unmittelbar aufeinander folgen. Hat man z.B. das Beispiel durch den Befehl

.i 634, 100

um 100 Plätze verschoben, so erhält man auf die Eingabe:

.p

die ("richtig nachgeführte") Antwort:

.p 634, 763

jedoch auf die Eingabe:
.p 634

die Antwort:
.p 634, 16788

(Je nach der Vorgeschichte kann hier auch eine andere Adresse als ADR2 erscheinen!)

9. Insert/Delete

. i ADR, n

mit n zwischen 0 und 32768 schafft in einem Maschinenprogramm, das in dem durch .p definierten Programmbereich steht, ab Adresse adr Platz für n Bytes. Dabei werden alle Operanden von JMP, JSR und relativen Sprungbefehlen korrigiert. Ist die Korrektur nicht möglich, z.B. weil die Sprungweite zu groß würde, wird das Programm nicht verändert und der betreffende Sprungbefehl mit "?" davor vom Disassembler ausgegeben.

. i ADR, -n

entfernt n Bytes ab Adresse ADR aus dem Programm und korrigiert alle Sprungbefehle. Hier kann eine Fehlermeldung dadurch entstehen, daß ein nicht gelöschter Sprung in den gelöschten Bereich zielt. Das Programm wird auch in diesem Fall nicht geändert.

Sprünge, die genau auf die Insert-Adresse ADR zielen, zeigen nachher hinter den eingefügten Bereich. Da dies nicht immer erwünscht ist, sollte man solche Sprünge nach der Verschiebung vorsichtshalber einzeln kontrollieren und nötigenfalls ändern.

Die INSERT-Funktion gestattet es auch, Programme im Speicher zu verschieben und dabei die Sprungadressen zu korrigieren. Zu diesem Zweck fügt man am Programmanfang die gewünschte (positive oder negative) Zahl von Bytes ein.

Achtung:

Falls mit dem Programm Datenbereiche verbunden sind, müssen diese jedoch gesondert behandelt werden; sie dürfen nicht in dem durch .p definierten Bereich liegen, da sie sonst in nicht vorhersehbarer Weise verändert werden können.

9. Insert / Delete

Beispiel:

Ermittlung des augenblicklichen Programmbereichs durch die Eingabe:

.p

Antwort:
.p 634, 662

Das Beispielprogramm soll nun im Speicher um 100 Plätze verschoben werden.

Achtung: Für Insert muß der definierte Bereich ausschließlich Programmbereich sein. Die Anfangsadresse in unserem Beispiel muß also unbedingt 650 lauten!

Neue Eingaben:
.p 650, 662

.i 650, 100

Antwort:
. 650 nop
. 651
. ...

Kontrolle des (nachgeführten) Programmbereichs (s.8.)..

Eingabe:
.p

Antwort:
.p 650, 762

9. Insert / Delete

Kontrolle des verschobenen Programms:

```
.    750  ldy    #15
.    752  lda    634,y
.    755  sta  33208,y
.    758  dey
.    759  bpl     -9    752
.    761  rts
.    762  brk
```

Achtung: Bei dem obigen INSERT-Befehl darf nicht i 634, 100 eingegeben werden, denn zwischen 634 und 650 steht kein Programm sondern Daten, diese müssten gesondert mit 'Transport' verschoben werden.

Man erkennt, daß die beiden Operanden der Befehle in den Zellen 752 und 755 unverändert sind. Beide Befehle beziehen sich auf Datenbereiche, sie werden in keinem Falle verändert!

Hier also nochmals der Hinweis, daß mit dem Programm verbundene Datenbereiche gesondert behandelt werden müssen!

Beispiel für Umrechnung von JSR- und JMP-Adressen:

Im Anschluß an das bisherige Programm werden ab Zelle 662 drei Sprungbefehle angefügt:

```
. 662 jsr 650
. 665 jmp 650
. 668 bne 650
. 670 rts
```

Der Programmbereich wird neu definiert mit:

Eingabe:
```
.p   650,   680
```

Es sollen nun 5 Plätze ab Zelle 650 eingefügt werden:

Eingabe:
```
.i 650, 5
```

Antwort:
```
.     650  nop
.     651  nop
.     652  nop
.     653  nop
.     654  nop
.     655  ldy     #15
.     657  lda     634,y
.     660  sta     33408,y
.     663  dey
.     664  bpl      -9     657
.     666  rts
.     667  jsr     655
.     670  jmp     655
.     673  bne     -20    655
.     675  rts
```

---

9. Insert / Delete

Wie man sieht, sind die Sprungadressen (auch die Adresse von bne !) um 5 Plätze umgerechnet worden. Den neuen (nachgeführten) Programmbereich erhält man nach der Eingabe :

.p

Antwort:
.p 650,685

Der Rücktransport geschieht in ähnlicher Weise nach:

.i650,-5

Antwort:

```
.   650  ldy   #15
.   652  lda   634,y
.   655  sta   666,y
.   658  dey
.   659  bpl    -9    652
.   661  rts
.   662  jsr   650
.   665  jmp   650
.   668  bne   -20    650
.   670  rts
```

Eingabe:
.p

Antwort:
.p 650, 680

10. Find

a) Suche im ROM

Zunächst muß eine Tabelle (siehe Anhang) angelegt werden, die die Aufteilung des ROM in Programm und Datenblöcke angibt. Diese Tabelle kann an beliebiger Stelle im RAM stehen:

A	Startadresse 1. Programmblock
A+2	Startadresse 1. Datenblock
A+4	Startadresse 2. Programmblock
...	
A+2n	Startadresse letzter Programmblock
A+2n+2	Adresse nach letztem Programmblock
A+2n+4	0 0

Mit der Tabellenstartadresse A in der Klammer (als Konstante oder Variable) können mit

.f(A),kode:operand

alle entsprechenden Befehlszeilen im ROM im Disassemblerformat gelistet werden.

Statt ´kode´ können mehrere Kodes durch Komma getrennt angegeben werden. Soll nur der Operand geändert werden, so kann die Angabe der Kodes entfallen: .f(A):operand sucht nach denselben Kodes wie der letzte .f(A) mit Kodeangabe. Statt operand kann ein Operandenbereich im Format operand-operand angegeben werden. Ist der Operand kein Suchkriterium, so kann die Angabe :operand entfallen.

10. Find

Damit nicht immer wieder viele Kodes eingegeben werden müssen, können anstelle von ´kode´ folgende Sondersymbole benützt werden:

S, SI, SII — für alle nicht-indizierten bzw. absolut-indizierten bzw. indirekt-indizierten Speicherbefehle (das sind alle Befehle, die den Inhalt einer Speicherzelle verändern, also auch ASL, INC usw.),

L, LI, LII — für alle entsprechenden Ladebefehle (das sind alle Befehle die den Inhalt einer Speicherzelle abfragen, also auch ADC, EOR usw.).

Nach einem Sondersymbol können weitere Einzelkodes, jedoch keine weiteren Sondersymbole eingegeben werden. Für die Operandenangabe gelten dieselben Regeln wie beim Normalformat.

b) Suche in eigenen Programmen:

Auch hier legt man ab einer beliebigen Adresse A eine Tabelle an, die die Aufteilung des Programms in Programm und Datenblöcke beschreibt und verfährt dann wie oben. Im einfachsten Fall (nur ein Programmblock) besteht die Tabelle aus zwei Adressen und der Adresse Null. Alle beschriebenen Sonderformen sind auch hier möglich.

Beispiel für Find im ROM-Bereich

(die Rechnerantworten beziehen sich auf den Typ 3032)

Zunächst muß für diese Funktion erforderliche Adressentabelle in einen beliebigen Speicherbereich abgelegt werden. Die für Ihren Rechner zutreffende Tabelle finden Sie im Anhang. Falls Sie diese Tabelle im Kassettenpuffer anlegen wollen, so beachten Sie bitte, daß SM-KIT die Adressen 919 bis 1000 lesend und schreibend verwendet. Als Tabellenanfang haben wir hier die Adresse 700 gewählt. Mit dem Befehl .700a ADR (siehe Abschnitt 5) können Sie die Tabelle eintipppen, danach können Sie die Funktion find im ROM anwenden.

Suche SEI :

```
. f(700),120
```

Antwort:

```
. 51486   sei
. 52845   sei
. 58019   sei
. 58040   sei
. 58687   sei
. 58813   sei
  (usw.)
```

Suche LDA 5

```
.f(700),169:5
```

Antwort:

```
. 55821   lda 5
. 62169   lda 5
. 63268   lda 5
. 64967   lda 5
```

---

10. Find

Suche JMP 112
.f(700),76:112

Antwort:
. 50964 jmp 112
. 52640 jmp 112
. 52736 jmp 112

Beipiel für Find im eigenen Maschinenprogramm

Für die Suche im eigenen Programm muß für Find an einer beliebigen Stelle des Speichers eine Adresstabelle erstellt werden. Hier wurde bei Speicherzelle 826 begonnen:

```
.    826   a    650    (Beginn des Programmbereichs)
.    828   a    662    (Ende des Progr.bereichs)
.    830   a      0    (Ende der Tabelle)
```

Suche RTS

Eingabe:
.f(826),96

Antwort:
. 661 rts

Suche den Befehl LDY# mit den Operanden 15 bis 20

Eingabe:
.f(826),160:15-20

Antwort:
. 650 ldy #15

---

10. Find

Suche Befehle, die den Inhalt einer Speicherzelle lesen:

Eingabe:
.f(826),l

Antwort:
.   (nichts gefunden !)

Eingabe:
.f(826),li

Antwort:
.   652   lda   634,y

Eingabe:
.f(826),lii
Antwort:
.   (nichts gefunden !)

Suche Befehle, die den Inhalt einer Speicherzelle verändern:

Eingabe:
.f(826),s

Antwort:
.   (nichts gefunden !)

Eingabe:
.f(826),si

Antwort:
.   655   sta 33408,y

Eingabe:
.f(826),sii
Antwort:
.   (nichts gefunden !)

---

11. Trace

Allgemeine Bemerkungen

Mit der Funktion TRACE kann der Ablauf eines Maschinenprogramms schrittweise abgearbeitet werden, die verschiedenen Registerinhalte werden dabei nach jedem Schritt aktualisiert auf dem Bildschirm ausgegeben. Der Aufsetzpunkt im Programm und der Trace-Beginn können (wie auch im Basic-Teil) frei vorgegeben werden. Während des 'tracens' überwacht das Maschinenprogramm 'SM-KIT' ein anderes Maschinenprogramm. Beide benötigen hierbei die ZERO-Page und das Stack. Damit dieses möglich wird, muß für SM-KIT eine neue ZERO-Page definiert werden (im folgenden Arbeitsbereich genannt) in die die ursprüngliche ZERO-Page ausgelagert wird. Dieser Arbeitsbereich hat eine Länge von 768 bytes und kann frei gewählt werden. Die Anfangsadresse dieses Arbeitsbereiches wird beim Einschalten des TRACE mit dem Befehl .PF (Abkürzung für Pfeil nach rechts) mit 10000 (dezimal) vorbesetzt, sie kann durch Überschreiben geändert werden.

Einschalten von Trace:

.> 10000 (Punkt, Pfeil nach rechts)

Drückt man nun die RETURN-Taste, so erscheint die erste Trace-Zeile:

. aaaxxxyyysss--++---- ADR

und darunter eine Erklärungszeile. Im Interesse der besseren Lesbarkeit erscheinen die Registerinhalte abwechselnd RVS und in Normaldarstellung, da Zwischenräume auf dem 40-spaltigen Bildschirm keinen Platz mehr haben. Die ersten 12 Stellen nach dem Punkt geben die Inhalte des ACCUs, X-Registers, Y-Registers und des Stacks wieder, (sie sind mit 0, der Stack ist mit 250 (Wert nach CLR) vorbesetzt), dahinter werden die verschiedenen Statusflags ausgegeben.

11. Trace

Nach RETURN wird ab der in der ersten Trace-Zeile vorgegebenen oder überschriebenen Adresse ADR der Trace-Lauf gestartet. Der Lauf wird solange fortgesetzt wie irgendeine Taste gedrückt ist.

Änderungen von Vorgaben:

Alle Zeichen in der Ausgabezeile können durch Überschreiben geändert werden; dabei sind natürlich die Beschränkungen für Bytes und Adressen zu beachten. Bei den Statusflags bedeutet + gesetzt, - gelöscht. Die RVS-Darstellung kann beim Überschreiben ignoriert werden Wird RETURN gedrückt, so wird der an der Trace-Adresse stehende Befehl disassembliert ausgegeben, dann ausgeführt. In der nächsten Zeile werden die neuen Registerinhalte und die neue Adresse ausgegeben.

Wird der Befehlskode vom Disassembler nicht erkannt, so wird "???" ausgegeben und der Kursor geht an den Anfang der Adresse zurück, damit diese geändert werden kann.

Programmabschnitte tracen:

Eingabe von "- ADR" nach der ausgegebenen Adresse (der Rest der Zeile muß gelöscht werden!, oder nach -ADR ein ´:´) und RETURN läßt das Programm bis zu der angegebenen Adresse ohne Ausgabe laufen. Wegen der Adressenüberwachung ist die Laufzeit dabei wesentlich erhöht. Mit der STOP-Taste kann dieser Modus jederzeit unterbrochen werden.

Eine Variante der Möglichkeit ´-ADR´ ist das Fortlassen der Angabe adr (also **nur** Bindestrich hinter der ausgegebenen Adresse eingeben und Rest der Zeile löschen!), man erreicht hierdurch einen völlig freien (unüberwachten) Ablauf. Diese Möglichkeit ist nur dort sinnvoll, wo man sicher weiß, daß das kommende Programm(-stück) ohne Fehler ist und zu einem sinnvollen Abschluß kommt! Andernfalls wird sich der Rechner nicht zurückmelden!

---

11. Trace

JSR-Unterprogramm tracen:

Nach Ausführung eines JSR-Befehls kann hinter der Adresse in der nächsten Zeile "- R" eingegeben werden (hier kann der Zeilenrest stehenbleiben). Das Programm läuft dann bis zum zugehörigen RTS im Normaltempo ab. Anschließend meldet sich die Trace-Routine mit dem aktuellen Registerstand an der auf den JSR-Befehl folgenden Adresse zurück. Die Eingabe "- R" muß nicht unmittelbar nach dem JSR-Befehl, sondern kann auch später noch gemacht werden, solang der Stack nicht verändert wurde (z.B. durch PHA)! Versucht man es trotzdem, so wird sich der Rechner in der Regel nicht zurückmelden.

Verlassen von Trace:

Man kann die Trace-Zeile jederzeit verlassen, z.B. um Speicherinhalte zu betrachten, zu ändern usw. Dabei ist zu beachten, daß die Speicherzellen 0,...,633 vor und nach jedem Zugriff mit dem Arbeitsbereich ausgetauscht werden. Wenn also der Arbeitsbereich bei 10000 beginnt und man wissen möchte, was das im Tracelauf untersuchte Programm in der Speicherzelle 17 vorfindet, muß man in 10017 nachsehen!

Bitte beachten Sie den Zusammenhang zwischen TRACE und BREAK-Point (siehe nächstes Kapitel!).

Vorsicht:

Wird der Arbeitsbereich nicht vorher mittels . > definiert (z.B. wenn ein BRK erreicht wird bevor definiert wurde), so beginnt er zwangsweise an der voreingestellten Adresse 10000! Wenn dieser Bereich nicht frei ist, sondern z.B. in das zu untersuchende Programm hineinreicht, wird es Ärger geben.

Beispiel:

In der TRACE-Funktion sind eine Reihe von Varianten integriert, diese sollen nacheinander an Hand des Beispiels gezeigt werden.

1. Trace-Lauf ohne Überwachung

Für die Behandlung von TRACE wird unser Programmbeispiel allgemeiner, wenn die Adresse von STA in Zelle 655 von 33408 in 666 abgeändert wird. Bitte stellen Sie das Beipiel durch .650 und kurzes Niederhalten von RETURN auf dem Bildschirm dar. Dann gehen Sie mit dem Cursor in die Zeile 655 und tippen ab der Spalte, wo jetzt noch die Adresse 33408 steht 666,Y ein, anschließend löschen Sie den Rest der Zeile und geben RETURN ein. Nun können Sie den Bildschirm löschen. Nach der Befehlseingabe . antwortet der Rechner mit 10000. Dies ist die voreingestellte Adresse, ab der der Rechner die Pages 0..2 für den TRACE-Lauf auslagert. Sollte dieser Bereich bei Ihnen schon belegt sein, so können Sie durch Überschreiben der Adresse 10000 auch einen anderen für Sie geeigneten Bereich definieren.

Nachdem Sie nun erneut RETURN eingetippt haben, ändern Sie die vom Rechner nach den Registerinhalten ausgegebene Adresse in 650 um und quittieren mit RETURN. Sie erhielten bisher folgende Bildschirmdarstellungen:

---

11. Trace

Eingabe:
.>

Antwort:
.>10000

Eingabe: RETURN

Antwort:

```
.>    0  0  0250--++----  (bel.Adresse)
.     a  x  y  s nv bdizc
```

Eingabe: (Adresse 650, RETURN)

```
.>    0  0  0250--++----  650
.     a  x  y  s nv bdizc
```

Antwort:

```
.>    0  0   0250--++----   650 ldy   #15
.>    0  0 15250--++----   652 lda   634,y
.     a  x  y  s nv bdizc
```

Durch Niederhalten der RETURN-Taste können Sie nun das Beispiel Schritt für Schritt abarbeiten. Der an der TRACE-Adresse stehende Befehl wird jeweils disassembliert und in der nächsten Zeile werden die neuen Registerinhalte und die neue Adresse ausgegeben.

11. Trace

Beim Niederhalten der RETURN-Taste erhalten Sie auf dem Bildschirm folgendes Gesamtbild:

```
.  0    0    0 250  --++----   650 ldy   #15
.  0    0   15 250  --++----   652 lda   634,y
.  76   0   15 250  --++----   655 sta   666,y
.  76   0   15 250  --++----   658 dey
.  76   0   14 250  --++----   659 bpl    -9
.  76   0   14 250  --++----   652 lda   634,y
.  69   0   14 250  --++----   655 sta   666,y
.  69   0   14 250  --++----   658 dey
.  69   0   13 250  --++----   659 bpl    -9

                     .
                     .
                     .

.  75   0    2 250  --++----   652 lda   634,y
.  45   0    2 250  --++----   655 sta   666,y
.  45   0    2 250  --++----   655 sta   666,y
.  45   0    2 250  --++----   658 dey
.  45   0    1 250  --++----   659 bpl    -9
.  45   0    1 250  --++----   652 lda   634,y
.  77   0    1 250  --++----   655 sta   666,y
.  77   0    1 250  --++----   658 dey
.  77   0    0 250  --++--+-   659 bpl    -9
.  77   0    0 250  --++--+-   652 lda   634,y
.  83   0    0 250  --++----   655 sta   666,y
.  83   0    0 250  --++----   658 dey
.  83   0  255 250  +-++----   659 bpl    -9
.  83   0  255 250  +-++----   661 rts
.    a    x    y   s  nv bdizc
```

Die Stelle nach dem Punkt enthält auf dem Bildschirm den ´Pfeil nach rechts´ der Trace-Anweisung! Zwischen x, y und s und Status wurde je ein Blank eingefügt, um die Lesbarkeit zu verbessern. Auf dem Bildschirm ist sie durch RVS-Darstellung auch ohne diese Blanks gewährleistet.

Die Ausgabe nach Zelle 661 (RTS) kann bei Ihnen anders aussehen. Da zu diesem RTS während Trace kein entsprechendes JSR durchgeführt wurde, ist auf dem Stapel auch keine entsprechende Rücksprungadresse verfügbar, der Weiterlauf ist also undefiniert.

2. Trace-Lauf mit Ausgabe von Speicherinhalten

Der Trace gibt den Inhalt von Speicherzellen aus, wenn im oberen Bildschirmteil der entsprechende Speicherbereich disassembliert ist. Es sind alle Ausgabemöglichkeiten (Bytes, Text oder Programm, auch gleichzeitig!) zulässig. Die Ausgabe bewirkt eine erhöhte Laufzeit.

Im unteren Teil des Bildschirms kann gleichzeitig und ungestört von dem oben dargestellten Speicherbereich ein TRACE-Lauf ausgeführt werden.

Man kann dies dadurch erreichen, daß man nach der Darstellung des Speicherbereichs im oberen Bildschirmteil mit dem CURSOR in den unteren Bildschirmteil fährt und dort den TRACE-Lauf mit .> RETURN beginnen lässt. Hierauf folgt die Antwort .> 10000. Unmittelbar nach RETURN erfolgt nun die Auslagerung der Pages 0..2. Nach dieser Erläuterung soll das Beispiel-Programm mit TRACE und mit Speicherausgabe behandelt werden.

Man stellt das Beispiel (wie bei 'Disassembler' gezeigt) mit Hilfe des Befehls .650 und kurzes Niederhalten der RETURN-Taste bis zur Speicherzelle 663 auf dem oberen Bildschirmteil dar. In Zeile 655 sollte STA 666,Y stehen. In der Zeile darunter stellt man nun mit dem Befehl .666' den in den nächsten 16 Speicherzellen enthaltenen Text dar. Hier wird man zunächst nichts Sinnvolles finden. Deswegen füllt man mit 16 Sternen auf. Man kann den gleichen Speicherbereich gleichzeitig in Byte-Darstellung auf dem Bildschirm darstellen.

---

11. Trace

Dann fährt man mit dem Cursor in den unteren Bildschirmbereich (etwa Zeile 20) und startet mit dem TRACE-Befehl und Niederhalten von RETURN das Tracen. Wegen der Adressenüberwachung ist die Laufgeschwindigkeit jetzt gegenüber dem ersten Lauf kleiner. Während des Tracens erkennt man, daß in den 16 Adressen ab der Zelle 666, (die auf dem Bildschirm durch .666´ dargestellt wurden), während des TRACE-Laufs mit dem bekannten Text gefüllt werden. Der Lauf kann durch Loslassen der RETURN-Taste jederzeit unterbrochen werden.

Bildschirm bei Beginn des Trace-Laufs

```
.      634   'SMK/M2.-BEISPIEL
.
.      666   '****************
.      682   '
.      666   b    42   42   42   42   42
.      671   b    42   42   42   42   42
.      676   b    42   42   42   42   42
.      681   b    42    0    0    0    0

. 10000
.>     0   0   0250--++----   650 ldy   #15
.>     0   0  15250--++----   652 lda   634,y
.    a   x   y   s  nv  bdizc
```

Bildschirm nach Ende des Trace-Laufs

```
.     634  'SMK/M2.-BEISPIEL
.
.     666  'SMK/M2.-BEISPIEL
.     682  '
.     666  B   83   77   75   47   77
.     671  B   50   46   45   66   69
.     676  B   73   83   80   73   69
.     681  B   76    0    0    0    0

.  77    0   1250--++----   658 dey
.  77    0   0250--++--+-   659 bpl      -9
.  77    0   0250--++--+-   652 lda     634,y
.  83    0   0250--++----   655 sta     666,y
.  83    0   0250--++----   658 dey
.  83    0255250+-++----    659 bpl      -9
.  83    0255250+-++----    661 rts
.  83    0255252+-++----    257 and     (48),y
. a    x   y   s   nv bdizc
```

3. TRACE-Lauf <u>ohne</u> Ausgabe (mit Überwachung)

Die Ausgabe des TRACE-Laufs kann unterdrückt werden, wenn nach der ersten ausgegebenen Adresse des Trace-Laufs eine zweite Adresse angegeben wird. Bis zu dieser Adresse läuft dann Trace ohne Ausgabe, jedoch **mit** Überwachung (größere Laufzeit!), der Lauf kann mit Stop jederzeit unterbrochen werden!

11. Trace

Wir stellen nun unser Beispiel mit Hilfe des Befehls .650 und kurzes Niederhalten der RETURN-Taste bis zur Speicherzelle 661 auf dem oberen Bildschirmteil dar und machen darunter folgende neue Eingaben:

```
.   661  ldy   #15
.   663  lda   634,y
.   666  sta 33288,y
.   669  dey
.   670  bpl    -9    663
.   672  rts
```

Nun im unteren Bildschirmteil TRACE einschalten:
. 10000

Antwort:

```
.>   0  0   0250--++----  Adresse
.    a  x   y s nv bdizc
```

Eingabe:

```
                          650-661
```

Antwort:

```
.>   0  0   0250--++----650-661
.>83      0255250+-++----661 ldy #15
.    a x   y   s nv bdizc
```

Man erkennt also, daß der TRACE-Lauf bis 661 **ohne** Ausgabe (aber wegen der Überwachung nicht im Normaltempo!) durchläuft. In 661 kehrt dann SMK/M wieder zur normalen Trace-Ausgabe zurück.

---

3. TRACE ohne Überwachung und ohne Ausgabe

Diese Möglichkeit der TRACE-Behandlung funktioniert nur nach einem **JSR.**

Bevor dieser Sonderfall gezeigt werden kann, muß unser Beispiel folgendermaßen abgeändert werden:

```
.    661  nop
.    662  jsr     750
.    665  rts
```

Nun wird der Programmbereich von 650 bis 666 mit dem Befehl .t650,662,750 um 100 Speicherzellen transportiert und nach Eingabe von .750 und kurzes Niederhalten von RETURN auf dem Bildschirm dargestellt:

```
.    750  ldy    #15
.    752  lda     634,y
.    755  sta   33408,y
.    758  dey
.    759  bpl      -9    752
.    761  nop
```

Damit dieses Unterprogramm eine auf dem Bildschirm sichtbare Änderung ergibt, werden danach noch folgende Änderungen gemacht:

```
.    761  rts
.    755  sta 33168,y
```

Kommt man nun während des TRACE-Laufes (nach Eingabe von .> 10000 und Adressenangabe 650) an die Stelle, wo JSR disassembliert wird, so kann man hinter der in der Folgezeile ausgegebenen Adresse ein -R einfügen (Rest der Zeile braucht nicht gelöscht zu werden). Nach RETURN wird nun die Unterroutine ohne Überwachung (Normaltempo!) abgearbeitet, die Wirkung unseres Unterprogramms kann man auf dem Bildschirm verfolgen.

---

12. Break-Point

Man kann in ein im RAM-Speicher stehendes Programm einen "Breakpoint" einbauen und dadurch den Programmlauf an dieser Stelle unterbrechen. Dieses Verfahren ruft normalerweise den Monitor auf. Im SMK/M2.0 gibt es hierfür eine besondere Routine, die den Breakpoint automatisch wieder entfernt und an der Stelle, wo der Breakpoint gesetzt wurde, den Tracemodus einschaltet.

Der Breakpoint wird gesetzt durch:

. b ADR

Dabei ist zu beachten, daß an der angegebenen Adresse ein gültiger Op-Kode stehen muß.

Ruft man anschließend das Programm durch SYS... auf, so meldet es sich mit Ausgabe der Trace-Zeile an der Breakpointadresse und kann von da aus im Tracelauf untersucht werden.

Vorsicht: Wird der Arbeitsbereich nicht vorher mittels .> definiert (z.B. wenn ein BRK erreicht wird bevor definiert wurde), so beginnt er an der voreingestellten Adresse 10000! Wenn dieser Bereich nicht frei ist,sondern z.B. in das zu untersuchende Programm hineinreicht, wird es Ärger geben. Weiterhin haben Vorbesetzungen in der ZERO-Page bei BRK keinen Sinn, da erst unmittelbar beim Erreichen von BRK ausgelagert wird.

Beispiel:

Allgemein dient ein Breakpoint (ähnlich wie STOP in BASIC) dazu, Programme stückweise zu testen. SM-KIT/M bietet nun den Service, daß der Rechner nach dem Aufruf des vor diesem BRK liegenden Programmteils (durch SYS....) bis zu diesem Punkt arbeitet, an dieser Stelle den BRK-Point entfernt und den Trace Modus einschaltet.

12. Break-Point

Setzen eines Break-points:

An der Stelle 661, wo bisher in unserem Beispiel RTS stand, soll ein BRK eingebaut werden. Dies geschieht durch .b 661.

Zunächst stellen wir im oberen Bildschirmteil unser Programm dar, und tippen dann .b661 RETURN ein.

Nun rufen wir unser Programm nach CLR (Bildschirm löschen) mit SYS 650, automatisch wird jetzt bei Erreichen der Zelle 661 der BRK entfernt und TRACE eingeschaltet.

Bildschirm bei Beginn:

```
.   650  ldy   #15
.   652  lda    634,y
.   655  sta  33408,y
.   658  dey
.   659  bpl     -9   652
.   661  rts
.b 661
```

Nach Bildschirm-löschen eintippen:
SYS 650

Antwort:
SMK/M2.-BEISPIEL

```
.83    0  255  250  +-++----   661
. a    x   y    s   nv bdizc
```

Anmerkung: Sie können die Wirkung des Breakpointsetzens auf dem Bildschirm verfolgen, wenn Sie, bevor Sie den Befehl .b661 geben, TRACE mit dem Befehl .> RETURN RETURN einschalten. Unmittelbar nach RETURN nach dem Befehl .b661 wird auf dem Bildschirm in der Zelle 661 ein BRK sichtbar werden.

---

13. Ausgabe von Basic-Zeilenadressen

Achtung:

Aus Platzgründen konnte diese Routine bei 8000/4000'er Rechner nicht mehr im SM-KIT untergebracht werden. Man kann jedoch diese Funktion von Basic aus mit dem SYS Aufruf

SYS40941,ZNR

direkt erhalten, als Antwort erhält man in diesem Falle die Startadresse der entsprechenden Basic-Zeile.

Für Rechner der **Serie 3000** ist diese Funktion im SMK/M2.0 direkt integriert!

. z ZNR

gibt die Anfangsadresse der BASIC-Zeile Nr. ZNR und die Anfangsadresse der darauffolgenden Zeile aus, falls die Zeile existiert.

Existiert die Zeile nicht, so wird ZNR in die Nr. der nächstfolgenden Zeile abgeändert, bevor die Adressen ausgegeben werden.

Falls weder die Zeile ZNR noch eine Zeile mit einer größeren Nummer existiert, wird ZNR in 64000 geändert, die Adresse der zweiten Null nach Programmende sowie eine (bedeutungslose) 0 ausgegeben.

13. Ausgabe von BASIC-Zeilenadressen

Beispiel:

Um sicherzustellen, daß kein anderes Basic-Programm im Speicher vorhanden ist, geben Sie bitte **NEW** ein.

Antwort:
READY.

Nun wird in die BASIC-Zeile 10 ein REM eingegeben:

10REM

Antwort: (wegen SM-KIT/B!)
10REM
20

Nach der Eingabe von .z0 erhält man als Antwort:

.z 10, 1025, 1031
.

Hierbei bedeutet die erste Ziffer die Zeilennummer, die zweite Ziffer ist die Anfangsadresse dieser Basiczeile, die dritte Ziffer ist die Anfangsadresse der folgenden Basiczeile.

Bei Rechnern der Serien 4000 und 8000 erhalten Sie nach dem SYS-Aufruf SYS 40941,0 oder SYS 40941,10 die Antwort:

1025

14. Einbau von REM-Routinen

. tp,z ZNR

erzeugt im BASIC-Programm eine REM-Zeile mit der Nummer ZNR, die das Maschinenprogramm ab der Anfangsadresse des durch .p definierten Programmbereichs bis zum ersten Kode 0 enthält. Eine solche REM-Routine kann in einem BASIC-Programm mittels USER-LINK aufgerufen werden.

Näheres zu USER-LINK und REM-Routinen erfahren Sie auf Anfrage beim Sofwareverbund.

Beispiel

Zunächst muß der Programmbereich definiert werden, es soll unser erstes Beispiel als REM-Routine aufgebaut werden. Bitte beachten Sie, daß in 661 RTS steht. Es belegt den Speicherbereich von 650 bis 661, als Programm bereich muß demnach .p650,662 eingegeben werden.

```
.   650  ldy   #15
.   652  lda    634,y
.   655  sta 33408,y
.   658  dey
.   659  bpl     -9    652
.   661  rts
```

Eingabe:
.p650,662

Bitte beachten Sie, daß **im Programm kein Code 0 enthalten** sein darf (an dieser Stelle würde in jedem Falle abgebrochen!), jedoch **muß das Programm mit 0 enden!** Weiterhin darf die REM-Routine höchstens eine Länge von 250 Bytes haben und keine JSR- oder JMP Sprünge auf sich selbst enthalten.

13. Einbau von REM-Routinen

Unsere REM-Routine soll in der Basiczeile 10 entstehen. Dies wird durch folgende Eingabe erreicht:

Eingabe:
.tp,z10

Zur Kontrolle kann durch LIST10 der Aufbau der REM-Routine nachgewiesen werden.

Achtung: Beim Arbeiten in Basic ist wichtig, daß Sie eine REM-Routine nie mit RETURN übernehmen dürfen! (In einem Maschinenprogramm können alle Codes vorkommen, also auch nicht druckbare Zeichen, durch RETURN-Übernahme in Basic wird also ein hinter REM stehendes Maschinenprogramm zerstört.)

Eingabe:
Li10

Antwort:
 10 REMCLOSEPOS:PRINTFREFORLET- +
READY.

Mit Hilfe von .z10 kann nun die Startadresse der Basiczeile 10 festgestellt werden:

Eingabe: .z10

Antwort: .z 10, 1025,1085

Bitte beachten Sie, daß das Maschinenprogramm 5 Adressen hinter der ausgegebenen Zeilenanfangsadresse beginnt, es kann also aufgerufen werden durch:

SYS1025+5
Antwort:
SMK/M2.-BEISPIEL

ROM-Adressen für MP-Find

Rechner-Typen:

3000	40alt	40neu	8000
49384	45858	45858	45858
52220	48375	48375	48375
52256	48409	48409	48409
52643	48800	48800	48800
52648	48805	48805	48805
53385	48908	48908	48908
53389	48909	48909	48909
55496	48961	48961	48961
55542	49036	49036	49036
55813	49881	49881	48881
55818	49885	49885	49885
56511	51954	51954	51954
56526	52000	52000	52000
56861	52271	52271	52271
56926	52276	52276	52276
57004	52969	52969	52969
57050	53112	53112	53112
57207	53447	53447	53447
57215	53512	53512	53512
57428	53590	53590	53590
57484	53636	53636	53636
57532	53793	53793	53793
57593	53801	53801	53801
57617	54014	54014	54014
57622	54060	54060	54060
57783	54108	54108	54108
57822	54169	54169	54169
59128	54193	54193	54193
59242	54198	54198	54198
59391	54347	54347	54347
61622	54386	54386	54386
63341	54596	54596	54596
63344	54649	54649	54649

ROM-Adressen für Find

64769	55352	55352	55352
64785	55411	55411	55411
64992	56989	56989	56989
65045	56990	56990	56990
65457	56996	56996	56996
65472	57344	57344	57344
65517	58638	58410	57401
0	58880	58414	57419
	58891	58783	57830
	61650	58880	57858
	63404	59199	58417
	63407	61650	58434
	64844	63404	58870
	65427	63407	58880
	65517	64844	59089
	0	65427	61650
		65517	63404
		0	63407
			64844
			65427
			65517
			0

SM Softwareverbund-
Microcomputer GmbH
Scherbaumstraße 29
8000 München 83

Telefon: 089/6 37 12 11
Aut. Auftragsannahme
Telefon: 089/6 70 58 13